Network IPC: Socket

Laboratorio Sistemi Operativi

Aniello Castiglione
Email: aniello.castiglione@uniparthenope.it

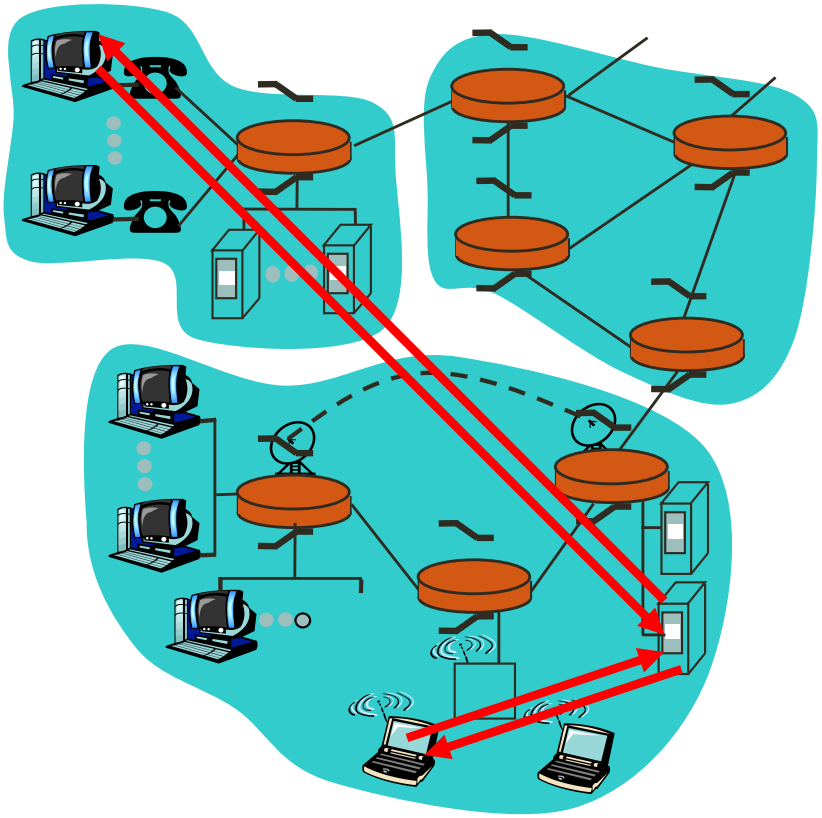
Processi comunicanti

- Due o più processi sullo stesso host comunicano usando una qualche forma di IPC definita dal SO
- Due o più processi su differenti host, connessi ad una rete di comunicazione comune, comunicano scambiandosi messaggi

Introduzione: architettura client/server

- Nei sistemi operativi moderni i servizi disponibili in rete si basano principalmente sul modello **client/server**
- Tale architettura consente ai sistemi di condividere risorse e cooperare per il raggiungimento di un obiettivo
- Per la programmazione di sistema, l'interfaccia delle **socket** fornisce un'astrazione user-friendly dei meccanismi di base per implementare programmi client/server

Architettura client-server



server sempre attivo che risponde alle richieste di servizi da parte dei client

server:

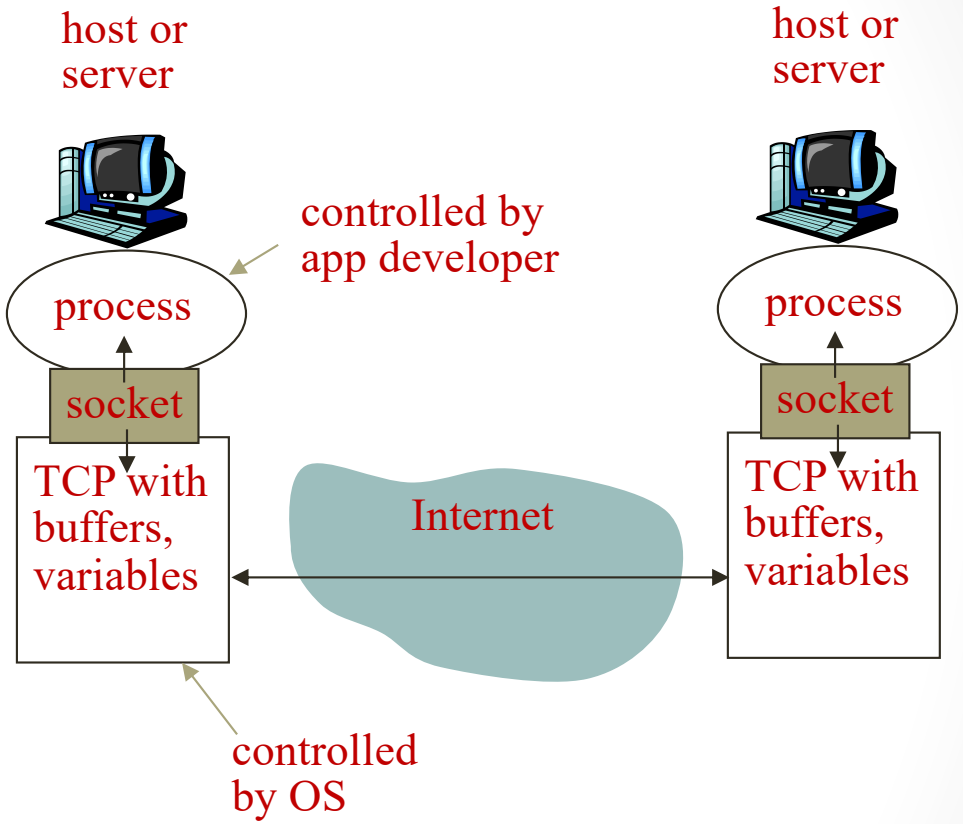
- host sempre attivo
- indirizzo IP fisso

client:

- comunica con il server
- può contattare il server in qualunque momento
- può avere indirizzi IP dinamici
- non comunica direttamente con gli altri client

Socket

- un processo invia/riceve messaggi alla/dalla sua socket
- una socket è analoga ad un "varco"
  - un processo che vuole inviare un messaggio, lo fa uscire dal proprio varco (socket)
  - il processo presuppone l'esistenza di un'infrastruttura esterna che trasporterà il messaggio attraverso la rete fino al varco del processo di destinazione



Indirizzamento

- Affinché un processo su un host invii un messaggio a un processo su un altro host, il mittente deve identificare il processo destinatario
- Un host A ha un indirizzo IP univoco a 32 bit
  - Ci chiediamo, è sufficiente conoscere l'indirizzo IP dell'host su cui è in esecuzione il processo per identificare il processo stesso?
    - No, sullo stesso host possono essere in esecuzione molti processi
  - L'identificatore comprende sia l'indirizzo IP che i numeri di porta associati al processo in esecuzione su un host
  - Esempi di numeri di porta:
    - HTTP server: 80
    - Mail server: 25

Servizi dei protocolli di trasporto

Servizio di TCP

- *orientato alla connessione:* è richiesto un setup fra i processi client e server
- *trasporto affidabile* fra i processi d'invio e di ricezione
- *controllo di flusso:* il mittente non vuole sovraccaricare il destinatario
- *controllo della congestione:* "strozza" il processo d'invio quando le rete è sovraccaricata

Servizio di UDP

- trasferimento dati *inaffidabile* fra i processi d'invio e di ricezione
- non offre: setup della connessione, affidabilità, controllo di flusso, controllo della congestione

Programmazione tramite socket

- Le applicazioni di rete, dunque, consistono di una coppia di programmi, il client ed il server che risiedono su sistemi differenti
- Quando questi due programmi vengono eseguiti, si crea un processo client e un processo server che comunicano tramite socket
- Quando si crea un'applicazione di rete, compito primario è la scrittura del codice per il client e per il server

Programmazione delle socket con TCP

Il client deve contattare il server

- Il processo server deve essere in corso di esecuzione
- Il server deve avere creato una socket per il benvenuto al contatto con il client

Il client contatta il server:

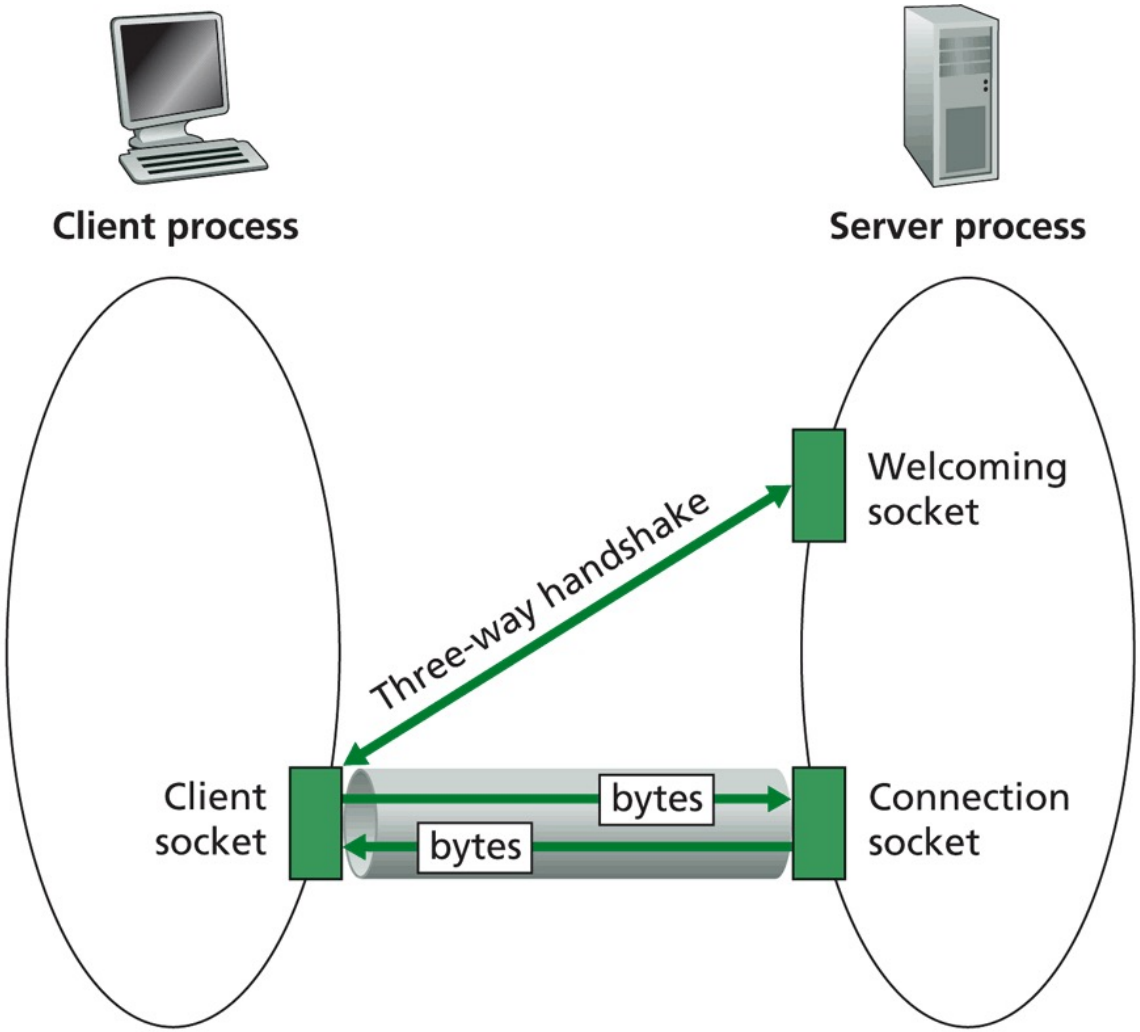
- Creando una socket TCP
- Specificando l'indirizzo IP, il numero di porta del processo server
- Quando il client crea la socket: il client TCP stabilisce una connessione con il server TCP (handshake a tre vie)

Quando viene contattato dal client, il server TCP crea una nuova socket per il processo server per comunicare con il client

- Consente al server di comunicare con più client
- Numeri di porta origine usati per distinguere i client

Dal punto di vista dell'applicazione, TCP fornisce un trasferimento di byte affidabile e ordinato tra client e server

Connessione socket TCP



Programmazione socket con UDP

- Non c'è "connessione" tra client e server
  - Non c'è handshaking
  - Il mittente allega esplicitamente ad ogni pacchetto l'indirizzo IP e la porta di destinazione
  - Il server deve estrarre l'indirizzo IP e la porta del mittente dal pacchetto ricevuto
  - i dati trasmessi possono perdersi o arrivare a destinazione in un ordine diverso da quello d'invio

Dal punto di vista dell'applicazione, UDP fornisce un trasferimento inaffidabile di gruppi di byte ("datagrammi") tra client e server

Socket

Programmazione di sistema

Domini della comunicazione

- Le socket sono create nell'ambito di un dominio di comunicazione che determina
  - Il metodo per identificare una socket
    - Il formato di un indirizzo di socket
  - Il range della comunicazione
    - Tra applicazioni sullo stesso host oppure tra host diversi connessi da una rete

Socket

- Esistono due modi principali per comunicare in rete:
  - il connection oriented model
  - il connectionless oriented model
- In corrispondenza dei due paradigmi di comunicazione precedenti abbiamo i seguenti tipi di socket:
- Stream socket: forniscono stream di dati affidabili, duplex, ordinati
  - Nel dominio Internet sono supportati dal protocollo TCP (Transmission Control Protocol)
- Socket a datagrammi: trasferiscono messaggi di dimensione variabile, preservando i confini ma senza garantire ordine o arrivo dei pacchetti
  - Supportate nel dominio Internet dal protocollo UDP (User Datagram Protocol)

Connessioni Socket (ProtocolloTCP)

- In primo luogo, un'applicazione server crea una socket che, come un descrittore di file, è una risorsa assegnata al processo server e solo a quel processo
  - Il server crea una socket mediante la system call socket() e non può essere condivisa con altri processi
  - Il processo server associa un nome alla socket
    - Alle socket locali si associa un filename nel file system
    - Per le socket di rete, il nome corrisponde ad un identificatore di servizio (numero di porta/ punto di accesso) rilevante per la particolare rete a cui i client possono connettersi

Connessioni socket (ProtocolloTCP)

- L'identificatore consente al sistema operativo di instradare le connessioni in arrivo specificando un nome particolare di porta al corretto processo server
- Il nome è assegnato alla socket usando la system call bind()
- Il processo server poi aspetta che un client si connetta alla socket a cui è stato dato il nome
  - La system call listen() crea una coda di connessioni in arrivo
  - Il server le può accettare usando la system call accept()

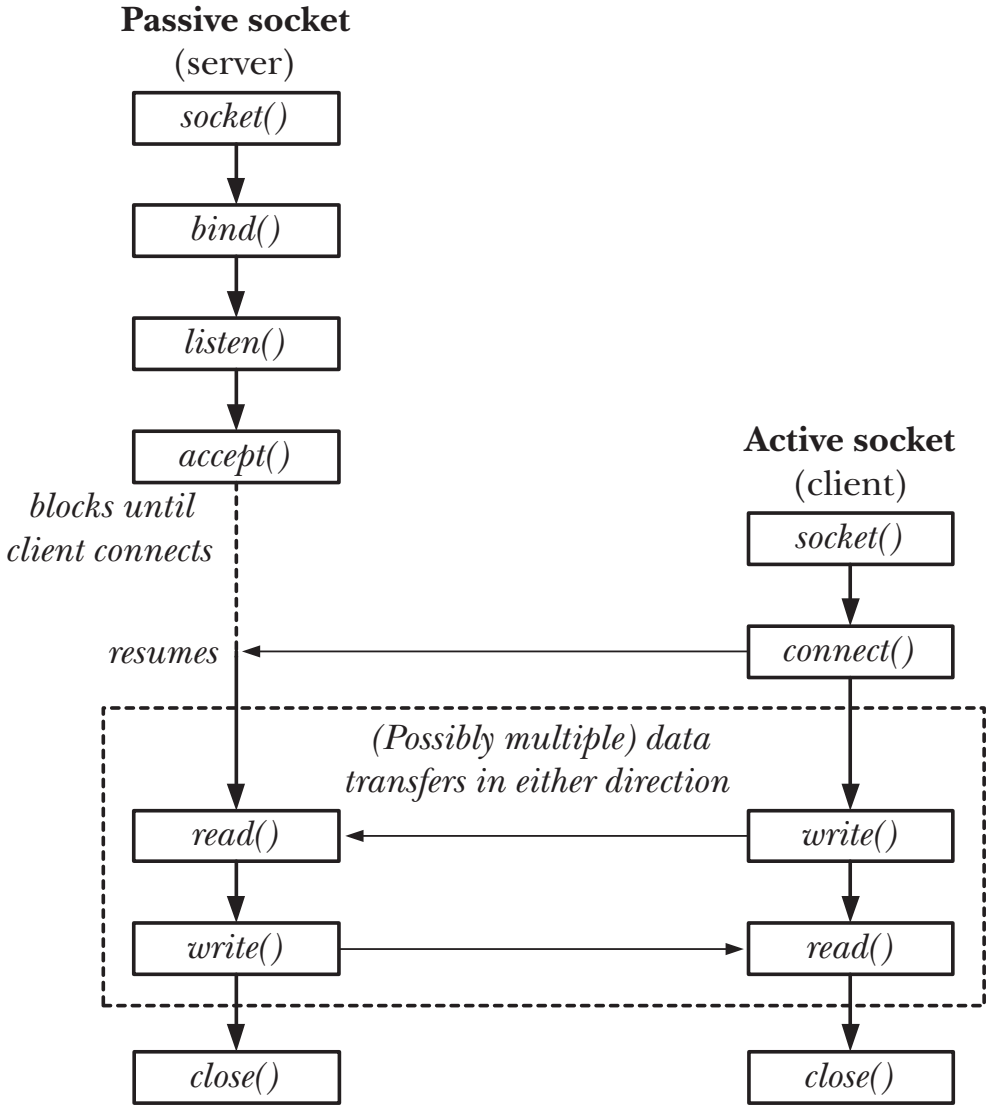
Connessioni Socket (ProtocolloTCP)

- Quando il server invoca accept(), viene creata una nuova socket distinta da quella dotata di nome (*three-way handshake*)
  - Questa nuova socket è usata solamente per comunicare con il client specifico
  - La socket dotata di nome resta a disposizione per ulteriori connessioni con altri client
  - Se il codice del server è scritto in modo appropriato, esso può trarre vantaggio da connessioni multiple
  - Con un server semplice, invece, altri client attendono sulla coda listen() fino a che il server non è di nuovo pronto

Connessioni socket (ProtocolloTCP)

- Il lato client di un sistema basato su socket è più diretto
- Il client crea una socket invocando socket()
- Successivamente invoca connect() per stabilire una connessione con il server usando la socket del server come indirizzo
- Una volta create, le socket possono essere usate come dei descrittori di file a basso livello, fornendo una comunicazione dati a due vie

Connessioni socket (ProtocolloTCP)



Un semplice client locale

- Vediamo un esempio di un semplice programma client con socket
  - Esso crea una socket priva di nome e la connette ad una socket server chiamata *server_socket*

Un semplice client locale

```
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <sys/un.h>
#include <unistd.h>
int main () {
  int sockfd;
  int len;
  struct sockaddr_un address;
  int result;
  char ch ='A';
  /* creiamo una socket per il client */
  sockfd = socket(AF_UNIX, SOCK_STREAM, 0);
```

Un semplice client locale

```
/* definiamo le generalità della socket del server */
  address.sun_family = AF_UNIX;
  strcpy(address.sun_path, "server_socket");
  len = sizeof(address);
/* connettiamo la nostra socket con quella del server */
  result = connect(sockfd, (struct sockaddr*)&address, len);
  if (result ==-1) {
       perror("ops:client 1");
       exit(1);
  }
  /* possiamo leggere e scrivere via sockfd */
  write(sockfd, &ch, 1);
  read(sockfd, &ch,1);
  printf("char dal server = %c\n",ch);
  close(sockfd);
  exit(0);
}
```

Un semplice client locale

- Quando eseguiamo il programma otteniamo un errore poiché non abbiamo ancora creato la socket lato server (il messaggio di errore può variare a seconda del sistema)

```
$ ./client1
Ops: client 1: Connection refused
$
```

Un semplice server locale

- Vediamo un semplice server locale che accetta le connessioni dal nostro client

- Esso crea la socket del server (socket()), gli assegna (bind()) un nome, crea una coda in ascolto (listen()), e accetta le connessioni (accept())

Un semplice server locale

```
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <sys/un.h>
#include <unistd.h>
#include <stdlib.h>
int main() {
  int server_sockfd, client_sockfd;
  int server_len, client_len;
  struct sockaddr_un server_address;
  struct sockaddr_un client_address;
  /* Rimuoviamo eventuali vecchie socket e creiamo
  una socket senza nome per il server */
  unlink("server_socket");
  server_sockfd = socket(AF_UNIX,SOCK_STREAM,0);
```

Un semplice server locale

```
/* Assegnamo un nome alla socket */
  server_address.sun_family = AF_UNIX;
  strcpy(server_address.sun_path, "server_socket");
  server_len = sizeof(server_address);
  bind(server_sockfd, (struct sockaddr*)&server_address,
  server_len);
/* creiamo una coda di connessione ed attendiamo i client */
  listen(server_sockfd, 5);
  while(1){
       char ch;
       printf("server in attesa\n");
       /* accetta una connessione */
       client_len = sizeof(client_address);
       client_sockfd = accept(server_sockfd, (struct
       sockaddr*)&client_address,&client_len);
```

Un semplice server locale

```
  /* Leggiamo e scriviamo al client su client_sockfd */
  read(client_sockfd, &ch,1);
  ch++;
  write(client_sockfd, &ch,1);
  close(client_sockfd);
  }
}
```

- Il programma server, in questo esempio, può solo servire un client alla volta
  - Legge un carattere dal client, lo incrementa, e lo riscrive
- In sistemi più sofisticati, in cui il server deve eseguire più lavoro per conto del client, ciò non sarebbe accettabile, poiché altri client non sarebbero in grado di connettersi fino a che il server non ha finito

Clint-Server locali in esecuzione

- Quando si esegue il programma server
  - Esso crea una socket ed attende le connessioni
  - Se si avvia il programma server in background, possiamo avviare i client in foreground

```
$ ./server1 &
[1] 1094
$ server in attesa
```

- Mentre attende le connessioni, il server stampa un messaggio
  - Nell'esempio precedente, il server crea una socket nel file system che è possibile visualizzare con il comando ls

```
$ ls -lF server_socket
srwxr-xr-x 1 staiano staiano 0 2007-05-21 13:42 server_socket=
```

- Il tipo socket è riconosciuto dalla s all'inizio dei permessi e da = alla fine. La socket è stata creata come un file ordinario, con i permessi modificati dall'umask corrente

Clint-Server locali in esecuzione

- Eseguendo il programma client riusciamo a connetterci con il server. Poiché la socket di server esiste, possiamo connetterci ad esso e comunicare

```
$ ./client1
server in attesa
char dal server = B
$
```

- L'output dal server e dal client si intrecciano sul terminale
  - E' possibile comunque vedere che il server ha ricevuto un carattere dal client, lo ha incrementato, e restituito. Il server continua ed attende il prossimo client
  - Eseguendo numerose volte i client insieme, questi saranno serviti in sequenza, sebbene l'output possa apparire intrecciato

Attributi delle socket

- Per comprendere le system call concernenti le socket è necessario entrare più nel dettaglio del networking di UNIX
- Le socket sono caratterizzate da tre attributi: *dominio*, *tipo* e *protocollo*
  - Esse hanno anche un indirizzo usato come nome
  - Il formato degli indirizzi varia a seconda del dominio, noto anche come *famiglia di protocolli*
  - Ciascuna famiglia di protocolli può usare una o più famiglie di indirizzi per definire il formato dell'indirizzo

Domini delle socket

- I domini specificano il mezzo della rete che la comunicazione socket userà
- Il più comune dominio socket è AF_INET, che si riferisce all'Internet Networking, usato su molte reti locali Linux e da Internet stessa
  - Il protocollo sottostante, Internet Protocol (IP), che ha solo una famiglia di indirizzi, impone un particolare modo di specificare i computer su una rete
    - L'indirizzo IP
- Sebbene i nomi si riferiscano quasi sempre a macchine sulla rete Internet, questi sono tradotti in indirizzi IP di basso livello
  - Un esempio di indirizzo IP è 192.168.1.99
  - Tutti gli indirizzi IP sono rappresentati da 4 numeri, ognuno minore di 256
  - Quando un client si connette attraverso la rete mediante una socket, esso necessita dell'indirizzo IP del server

Domini delle socket

- Possono esserci numerosi servizi su un server. Un client può indirizzare un particolare servizio su una macchina usando una **porta IP**
  - Nel sistema una porta è identificata internamente da un intero unico a 16 bit ed esternamente dalla combinazione dell'indirizzo IP e del numero di porta
  - Le socket sono gli estremi della comunicazione che devono essere legate alle porte prima che avvenga la comunicazione

Domini delle socket

- I server aspettano le connessioni su porte specifiche. A servizi noti sono allocati numeri di porta usati da tutte le macchine Linux e Unix
  - Solitamente sono numeri inferiori a 1024
  - rlogin(513), ftp(21), http (80)

- Poiché c'è un insieme standard di numeri di porta per i servizi standard, i computer possono collegarsi facilmente tra loro senza dover stabilire il numero di porta corretto
  - I servizi locali possono usare indirizzi di porta non standard

Domini delle socket

- Il dominio nell'esempio del Client-Server locale, è il dominio del file system UNIX, AF_UNIX, che può essere usato dalle socket sulla base di un singolo computer
  - In questo caso, il protocollo sottostante è il file di input/output e gli indirizzi sono nomi di file assoluti

- I domini specificati da POSIX.1 sono
  - AF_INET
  - AF_INET6
  - AF_UNIX
  - AF_UNSPEC

- Possono essere usati anche altri domini come AF_ISO, per reti basate su protocolli standard ISO e AF_XNS, per Xerox Network System

Tipi di socket

- Un dominio socket può avere diversi modi per comunicare, ognuno dei quali potrebbe avere caratteristiche differenti
  - Questo non è un problema con le socket di dominio AF_UNIX, le quali forniscono un'affidabile percorso di comunicazione a due vie
  - In domini di rete, tuttavia, è necessario sapere le caratteristiche della rete sottostante
- I protocolli internet forniscono due livelli distinti di servizio: *stream* e *datagram*

Socket stream

- Le socket stream forniscono una connessione che è un flusso di byte a due vie affidabile e sequenziato
  - E' garantito che i dati non siano persi, duplicati o riordinati senza un'indicazione dell'occorrenza di un errore
  - I messaggi più grandi sono frammentati, trasmessi, e riassemblati
  - E' come un flusso su file, poiché accetta grandi quantità di dati e li scrive su dischi di basso livello in blocchi più piccoli
- Le socket stream, specificate dal tipo SOCK_STREAM, sono implementate nel dominio AF_INET dalle connessioni TCP/IP

Socket Datagram

- In contrasto, una socket datagram, specificata dal tipo SOCK_DGRAM, non stabilisce né mantiene una connessione. C'è anche un limite sulla dimensione del datagrammma che può essere inviato
- E' trasmesso come un messaggio di rete singolo che può andar perso, duplicato o arrivare fuori sequenza
- Le socket datagram sono implementate nel dominio AF_INET dalle connessioni UDP/IP e forniscono un servizio non affidabile e non sequenziato
  - Tuttavia, esse sono poco costose in termini di risorse, poiché non devono essere mantenute le connessioni di rete
  - Sono veloci, perché non c'è associato un tempo di impostazione della connessione

Creare una socket

```
#include <sys/socket.h>
int socket(int dominio, int tipo, int protocollo);
/* restituisce il descrittore di file (socket)se OK,
  -1 in caso di errore */
```

- La system call socket() crea una socket e restituisce un descrittore che può essere usato per accedere alla socket stessa
- La socket creata è un'estremo del canale di comunicazione
  - Il parametro *dominio* specifica la famiglia di indirizzi
  - Il parametro *tipo* specifica il tipo di comunicazione da usare con questa socket
  - *protocollo* specifica il protocollo da impiegare

Creare una socket

- I domini possono essere:
  - AF_UNIX: UNIX (file system socket)
  - AF_INET: ARPA Internet Protocols (UNIX network sockets)
  - AF_ISO: ISO standard protocols (Non POSIX.1)
  - AF_NS: Xerox Network System Protocols (Non POSIX.1)
- I domini più comuni sono AF_UNIX, usato per socket locali implementate via i file system di UNIX e Linux e AF_INET, usato per le socket di rete UNIX
- Le socket AF_INET possono essere usate da programmi che comunicano attraverso una rete TCP/IP, inclusa Internet
  - L'interfaccia Windows Winsock fornisce accesso a questo dominio di socket

Creare una socket

- Il parametro *tipo* specifica le caratteristiche della comunicazione da usare per la nuova socket
  - Possibili valori sono
    - SOCK_STREAM
    - SOCK_DGRAM

Creare una socket

- Il *protocollo* usato per la comunicazione è determinato dal tipo di socket e dal dominio
  - Solitamente il valore è zero
    - Per selezionare il valore di default per i dati dominio e tipo
      - TCP per SOCK_STREAM nel dominio AF_INET
      - UDP per SOCK_DGRAM nel dominio AF_INET
- Se sono supportati più protocolli per lo stesso dominio e tipo di socket
  - l'argomento *protocollo* è usato per selezionare un particolare protocollo

Formati degli indirizzi

- Un indirizzo identifica un lato socket in un particolar dominio di comunicazione
- Il formato dell'indirizzo è specifico al dominio
  - Per poter fornire tali indirizzi alle funzioni per le socket, gli indirizzi subiscono un cast ad una struttura indirizzo generica

```
struct sockaddr {
  sa_family_t  sa_family;
  char         sa_data[];
  .
  .
  .
  }
```

Indirizzi delle socket

- Ciascun dominio richiede il proprio formato di indirizzo. Per una socket AF_UNIX, l'indirizzo è descritto da una struttura, sockaddr_un, definita nel file <sys/un.h>

```
struct sockaddr_un {
short sa_family;     /* Flag AF_UNIX */
char sun_path[];     /* Pathname */
};
```

- Per indicare una socket nel dominio AF_INET, l'indirizzo è specificato usando una struttura chiamata sockaddr_in, definita in <netinet/in.h>, che contiene almeno tre membri:

```
struct sockaddr_in {
short sa_family; /* Flag AF_INET */
short sin_port;  /* Numero di porta */
struct in_addr sin_addr; /* indir. IP */
};
```

dove **in_addr** rappresenta un indirizzo IP

```
struct in_addr { u_long s_addr; /* 4 byte */ };
```

Indirizzi delle socket

- I quattro byte di un indirizzo IP costituiscono un valore di 32 bit singolo. Una socket AF_INET è descritta totalmente dal suo dominio, l'indirizzo IP ed il numero di porta

- Da un punto di vista dell'applicazione, tutte le socket agiscono come descrittori di file e sono indirizzate da un unico valore intero

Assegnare un nome alle socket

- Per rendere disponibile una socket all'uso (dopo averla creata con socket()) ad altri processi, un programma server ha bisogno di assegnare un nome alla stessa
  - le socket AF_UNIX sono associate con un pathname del file system
  - le socket AF_INET sono associate con un numero di porta IP

```
#include<sys/socket.h>
int bind(int sockfd, struct sockaddr *address, size_t len);
/* restituisce 0 se OK, -1 in caso di errore */
```

- La system call bind() assegna l'indirizzo specificato nel parametro, *address*, alla socket senza nome associata con il descrittore di file *sockfd*. La lunghezza della struttura dell'indirizzo è passato come *len*

Creare una coda per la socket

- Per accettare le connessioni in arrivo su una socket, un programma server deve creare una coda per memorizzare le richieste pendenti

```
#include<sys/socket.h>
int listen(int sockfd, int backlog);
/* Restituisce 0 se OK, -1 in caso di errore */
```

- Questa funzione è utilizzata da un server per stabilire il massimo numero di connessioni da lasciare in coda
- Solitamente è eseguita dopo le chiamate di sistema socket() e bind() ed immediatamente prima della chiamata accept()
  - Imposta la lunghezza della coda a *backlog*

Accettare le connessioni

- Una volta che un programma server ha creato e “dato un nome” ad una socket, esso può aspettare le connessioni usando la system call accept():

```
#include<sys/socket.h>
int accept(int sockfd, struct sockaddr *address, size_t *len);
/* Restituisce il descrittore di file (socket)se OK, -1 in
   caso di errore */
```

- La system call accept() ritorna quando un programma client tenta di connettersi alla socket specificata dal parametro *sockfd*
- Il client è la prima connessione pendente nella coda della socket
  - La funzione accept () crea una nuova socket per comunicare con il client e restituisce il suo descrittore
  - La nuova socket avrà lo stesso tipo della socket in ascolto del server

Accettare le connessioni

- Alla socket deve essere stato precedentemente assegnato un nome da una chiamata a bind() ed allocata una coda da una chiamata a listen()
- L'indirizzo del client chiamante sarà posto nella struttura *sockaddr* puntata da *address*
- Il parametro *len* specifica la lunghezza della struttura del client
  - Se l'indirizzo del client è più lungo di questo valore, sarà troncato
  - Prima di chiamare accept(), *len* deve essere impostato alla lunghezza attesa dell'indirizzo
    - Al ritorno *len* sarà impostato alla lunghezza reale della struttura dell'indirizzo del client chiamante

Accettare le connessioni

- Se non ci sono connessioni pendenti sulla coda della socket, accept() blocca il processo fino a che un client effettua una connessione

- La funzione accept() restituisce
  - un descrittore della nuova socket quando c'è una connessione pendente di un client
  - -1 se si verifica un errore

Richiedere connessioni

- I programmi client si connettono ai server stabilendo una connessione tra una socket senza nome e la socket del server in ascolto. Il tutto è fatto invocando connect()

```
#include<sys/socket.h>
int connect(int sockfd, struct sockaddr *address, size_t len);
/* Restituisce 0 se OK, -1 in caso di errore */
```

- La socket specificata dal parametro *sockfd* è connessa alla socket del server specificata dal parametro *address*, che è di lunghezza *len*
- La socket deve essere un descrittore di file valido ottenuto da una chiamata a socket
- Se la connessione non può essere impostata immediatamente, connect() bloccherà il processo per un periodo di tempo (timeout) non specificato. Una volta trascorso timeout, la connessione sarà annullata e connect() fallisce
  - Tuttavia, se la chiamata a connect() è interrotta da un segnale gestito, la connect() fallirà, ma il tentativo di connessione non sarà annullato ma sarà impostato in modo asincrono

Chiudere una socket

- Si può terminare una connessione socket al server e al client invocando close() così come avviene per i descrittori di file

- E' necessario sempre chiudere la socket in ambo i lati

Esempio

- Sfruttiamo l'esempio visto all'inizio per creare una socket di rete piuttosto che una del file system
- Scegliamo il numero di porta 9734
  - Questa è una scelta arbitraria che evita i servizi standard (assegnati alle porte fino a 1024)
  - Nel file /etc/services sono elencati i servizi e le porte associate
- Eseguiremo il client ed il server attraverso una rete locale
  - Usiamo la rete di loopback che consiste di un singolo computer chiamato localhost con l'indirizzo standard 127.0.0.1, ovvero la macchina locale

Esempio: lato client

```
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <unistd.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdlib.h>
int main() {
  int sockfd;
  int len;
  struct sockaddr_in address;
  int result;
  char ch ='A';
/* creiamo una socket per il client */
sockfd = socket(AF_INET, SOCK_STREAM, 0);
```

Esempio: lato client

```
/* assegnamo un nome alla socket  */
   address.sin_family = AF_INET;
   address.sin_addr.s_addr = inet_addr("127.0.0.1");
   address.sin_port = 9734;
   len = sizeof(address);
/* connettiamo la nostra socket con quella del server */
  result = connect(sockfd, (struct sockaddr*)&address, len);
  if (result ==-1) {
       perror("ops:client 1");
       exit(1);
  }
  /* possiamo leggere e scrivere via sockfd */
  write(sockfd, &ch, 1);
  read(sockfd, &ch,1);
  printf("char dal server = %c\n",ch);
  close(sockfd);
  exit(0);
}
```

Esempio: lato client

- Il programma client usa la struttura *sockaddr_in* dal file netinet/in.h per specificare un indirizzo AF_INET
- Cerca di connettersi ad un server sull'host con indirizzo IP 127.0.0.1
  - Utilizza una funzione, inet_addr(), per convertire la rappresentazione testuale di un indirizzo IP in una forma adatta all'indirizzamento delle socket

```
in_addr_t  inet_addr(char *cp);
```

    - Converte una stringa di caratteri da una notazione con il punto decimale ad un indirizzo di Internet di 32 bit

Formati indirizzo

- Per la trasformazione inversa è possibile usare la funzione

```
char *inet_ntoa(struct in_addr in);
```

- Tuttavia inet_addr() e inet_ntoa() lavorano solo con indirizzi IPv4
- Per lavorare sia con indirizzi IPv4 che IPv6 è possibile usare la nuova funzione

```
#include<arpa/inet.h>
int inet_pton(int domain, const char * src, void *dst);
/* Restituisce 1 se OK, 0 se il formato non è valido, o -1 in caso
   di errore */
```

- inet_pton() converte una stringa di testo in un indirizzo binario nell'ordinamento di byte della rete

- Sono supportati solo due valori dell'argomento *domain*: AF_INET e AF_INET6

Esempio: lato server

```
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <unistd.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdlib.h>
int main() {
  int server_sockfd,client_sockfd;
  int server_len, client_len;
  struct sockaddr_in server_address;
  struct sockaddr_in client_address;
  /* crea un socket senza nome per il server */
  server_sockfd = socket(AF_INET,SOCK_STREAM,0);
```

Esempio: lato server

```
/* assegna un nome alla socket */
  server_address.sin_family = AF_INET;
  server_address.sin_addr.s_addr = inet_addr("127.0.0.1");
  server_address.sin_port = 9734;
  server_len = sizeof(server_address);
  bind(server_sockfd, (struct sockaddr*)&server_address, server_len);
  /* creiamo una coda di connessione ed attendiamo i client */
  listen(server_sockfd, 5);
  while(1){
    char ch;
    printf("server in attesa\n");
    /* accetta una connessione */
    client_len = sizeof(client_address);
    client_sockfd = accept(server_sockfd,(struct
  sockaddr*)&client_address,&client_len);
        /* Leggiamo e scriviamo al client su client_sockfd */
        read(client_sockfd, &ch,1);
        ch++;
        write(client_sockfd, &ch,1);
        close(client_sockfd);
        }
}
```

Esempio: lato server

- Il programma server crea un socket di dominio AF_INET e si dispone per accettare connessioni
- La socket viene legata alla porta scelta
  - L'indirizzo specificato determina a quali computer è consentito connettersi
  - Specificando l'indirizzo di loopback, come nel programma client, restringiamo le comunicazioni alla macchina locale

Ordinamento dei Byte

- Se si eseguono i programmi *server* e *client* è possibile vedere le connessioni di rete usando il comando netstat
  - Il comando mostra le connessioni client/server in attesa di chiusura
    - La connessione si chiude dopo un piccolo timeout

```
$ ./server2 &
[4] 1225
$ server in attesa
Client2
Server in attesa
Char dal server = B
$ netstat
Active Internet Connections
Proto Recv-Q Send-Q Local Address Foreign Address (State)  user
Tcp     0      0   localhost:1574 localhost:1174 TIME_WAIT root
```

Ordinamento dei Byte

- Possiamo osservare i numeri di porta che sono stati assegnati alla connessione tra il server ed il client
  - L'indirizzo locale (local address) mostra il server
  - L'indirizzo straniero (foreign address) è il client remoto
  - Per assicurare che tutte le socket siano distinte, queste porte client sono tipicamente differenti dalle socket del server in ascolto e uniche
- Tuttavia, all'indirizzo locale (la socket del server) è assegnata la porta 1574 sebbene abbiamo scelto la porta 9734
  - Perché differiscono?

Ordinamento dei Byte

- I numeri di porta e gli indirizzi sono comunicati sulle interfacce delle socket come numeri binari
  - Computer differenti impiegano ordinamenti differenti per gli interi
    - Un processore Intel memorizza l'intero a 32 bit come quattro byte di memoria consecutivi nell'ordine 1-2-3-4, dove 1 è il byte più significativo (little-endian)
    - I processori Motorola memorizzano l'intero nell'ordine di byte 4-3-2-1 (big-endian)
  - Se la memoria usata per gli interi è copiata byte per byte, i due computer hanno valori differenti assegnati all'intero

Ordinamento dei Byte

- Per consentire a computer di tipo differente di avere rappresentazioni degli interi, trasmessi in rete, coerenti è necessario avere un ordinamento nella rete
- I programmi client e server devono convertire le rispettive rappresentazioni interne degli interi nell'ordinamento della rete prima di effettuare la trasmissione
- Per questo si usano le funzioni:

```
#include <netinet.h>
usigned long int htonl(unsigned long int hostlong);
unsigned short int htons(unsigned short int hostshort);
unsigned long int ntohl(unsigned long int netlong);
unsigned short int ntohs(unsigned short int netshort);
```

Ordinamento dei Byte

- Queste funzioni convertono gli interi a 16 e 32 bit tra il formato host nativo e l'ordinamento standard della rete
  - Per i computer in cui l'ordinamento nativo coincide con l'ordinamento della rete queste rappresentano operazioni nulle
- In virtù di ciò, i codici client e server devono essere modificati opportunamente:
  - Server

```
server_address.sin_addr.s_addr = htonl(INADDR_ANY);
server_address.sin_port = htons(9734);
```

  - Client

```
address.sin_port = htons(9734);
```

- Il server è stato cambiato per consentire connessioni a qualsiasi indirizzo IP usando INADDR_ANY

send e recv

- Una volta stabilita la connessione fra client e server, si possono usare le system call send() e recv() per trasmettere e ricevere dati attraverso le socket:

```
#include <sys/types.h>
#include <sys/socket.h>
int send(int sockfd, void *buffer, int length, int flags);
int recv(int sockfd, void *buffer, int length, int flags);
```

- se *flags* vale 0, allora send() e recv() equivalgono, rispettivamente alle system call write() e read(). Altrimenti *flags* può assumere i seguenti valori, per quanto riguarda send():
  - MSG_OOB: il processo invia dati “out of band”;
  - MSG_DONTROUTE: vengono ignorate le condizioni di routing dei pacchetti sottostanti al protocollo utilizzato
- Per quanto riguarda recv() invece *flags* può assumere i seguenti valori:
  - MSG_PEEK: i dati vengono letti, ma non “consumati” in modo che una successiva recv() riceverà ancora le stesse informazioni
  - MSG_OOB: legge soltanto i dati “out of band”;
  - MSG_WAITALL: la recv() ritorna soltanto quando la totalità dei dati è disponibile
- Il valore di ritorno è la lunghezza del messaggio inviato/ricevuto

Esempio di applicazione connection oriented

- Realizziamo un “maiuscolatore". Un server che riceve delle stringhe di testo dai client, restituendole a questi ultimi dopo aver convertito in maiuscolo le lettere

```
/* upperserver.c: un server per rendere maiuscole linee di
  testo */
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <ctype.h>
#include <sys/times.h>
#define SERVER_PORT 1313
#define LINESIZE 80
```

Esempio di server: “maiuscolatore”

```
void upperlines(int in, int out) {
  char inputline[LINESIZE];
  int len, i;
  while ((len = recv(in, inputline, LINESIZE, 0)) > 0) {
       for (i=0; i < len; i++)
              inputline[i] = toupper(inputline[i]);
       send(out, inputline, len, 0);
  }
}
int main (int argc, char **argv) {
  int sock, client_len, fd;
  struct sockaddr_in server,client;
  /* impostazione dell'end point della comunicazione */
  if((sock = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
    perror("chiamata alla system call socket fallita");
    exit(1);
  }
  server.sin_family = AF_INET;
  server.sin_addr.s_addr = htonl(INADDR_ANY);
  server.sin_port = htons(SERVER_PORT);
```

Esempio di server: maiuscolatore

```
/* leghiamo l'indirizzo all'end point della comunicazione */
  if (bind(sock, (struct sockadrr*)&server, sizeof server)==-1) {
       perror("chiamata alla system call bind fallita");
       exit(2);
  }
  listen(sock, 1);
  /* gestione delle connessioni dei client */
  while (1) {
   client_len = sizeof(client);
   if ((fd = accept(sock, (struct sockaddr*)&client, &client_len))<0)
  {
         perror("accepting connection");
         exit(3);
  }
  fprintf(stdout, "Aperta connessione.\n");
  send(fd, "Benvenuto all'UpperServer!\n", 27, 0);
  upperlines(fd, fd);
  close(fd);
  fprintf(stdout, "Chiusa connessione.\n");
  }
}
```

Test del server

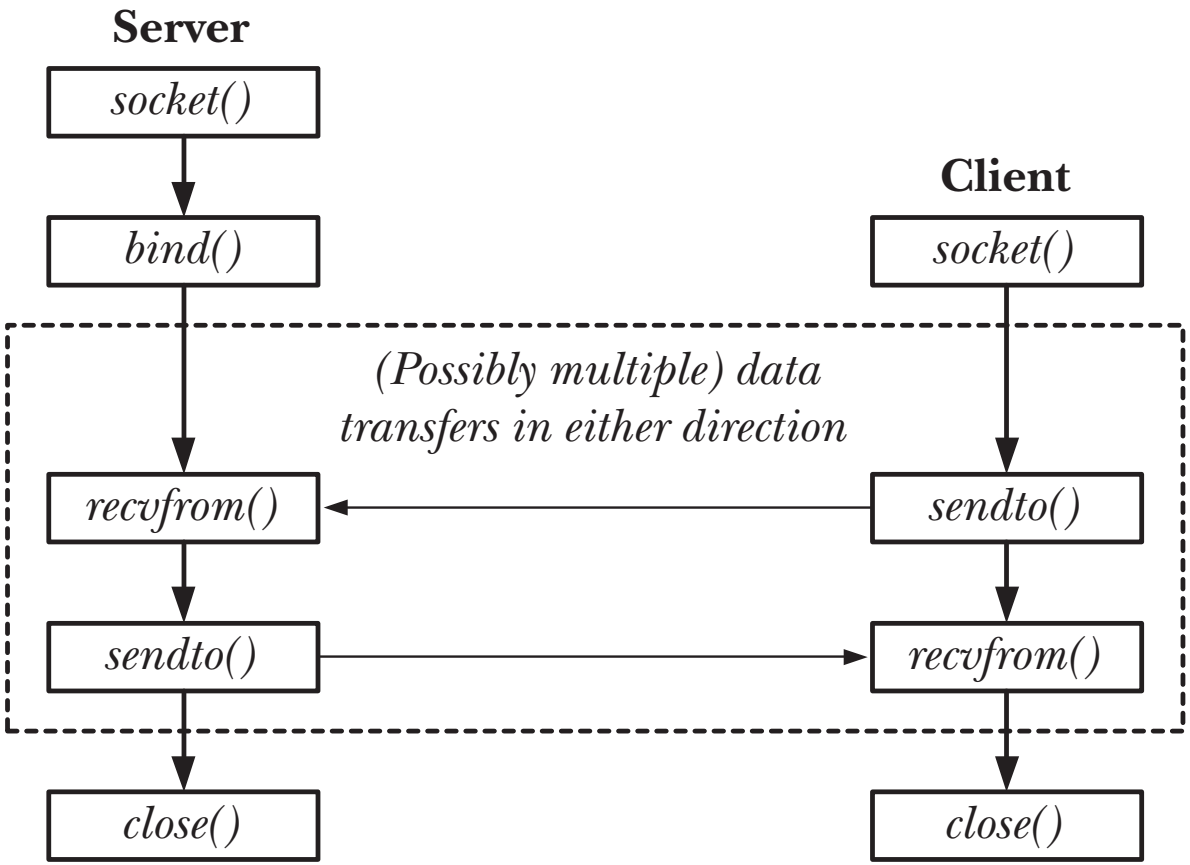
- Compiliamo e lanciamo in esecuzione il server:

```
$ gcc -o upperserver upperserver.c
$ ./upperserver &
```

- Per verificare il funzionamento del server, usiamo telnet:

```
$ telnet 127.0.0.1 1313
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
Benvenuto all'UpperServer!
prova
PROVA
Per terminare la sessione premere Ctrl + ] e
  impartire il comando quit:
^]
telnet> quit
Connection closed.
>
```

Connessioni socket (Protocollo UDP)



Esercizi

- Completare l'esempio del maiuscolatore, scrivendo il codice del client (caso del modello connection oriented). La struttura di quest'ultimo sarà la seguente:

```
#include <ctype.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
main() {
   int sockfd;
   if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror("chiamata alla system call socket fallita");
        exit(1);
   }
/* connessione al server */
/* invio e ricezione della stringa */
/* chiusura della connessione */
}
```

- Modificare il programma upperserver.c in modo che accetti più connessioni contemporaneamente (utilizzando la fork())